PL/SQL Collections

- Jayendra Khatod

Objectives

- Describe PL/SQL Collections
- Types of collections in PL/SQL
- Index By Table
- Nested Table
- VARRAY
- BULK COLLECT, FOR ALL Clauses
- Performance gains with Bulk Processing
- Array processing with BULK COLLECT and FORALL

Collections in PL/SQL

- **In PL/SQL a *collection variable* is a variable that can store zero, one ore more elements of a specific type (either an internal data type or a user defined data types). The type of the variable is itself a user defined type. Since a collection is a user defined type, a collection type can store collections as well.**

Different collection types

There are three collection types in PL/SQL:

- **Nested tables**
- **Index-by tables, also known as *associative arrays***
- **Varrays**

- **Nested tables extend the functionality of index-by tables. The main difference is that nested tables can be stored in a table column while index by tables can not.**

Example : Index by tables

```
declare type
    assoc_varchar is table of varchar2(10) index by pls_integer;
    var_assoc_varchar assoc_varchar;
    elem varchar2(10);
begin
    var_assoc_varchar(40) := 'forty';
    var_assoc_varchar(10) := 'ten';
    var_assoc_varchar(30) := 'thirty';
    var_assoc_varchar(20) := 'twenty';
    elem := var_assoc_varchar.first;
    while elem is not null
    loop
    dbms_output.put_line(elem || ': ' || var_assoc_varchar(elem));
    elem := var_assoc_varchar.next(elem);
    end loop;
end;
```

Example : Nested tables

```
declare
    type table_varchar is table of varchar2(10);
    var_table_varchar table_varchar;
begin
    var_table_varchar := table_varchar ('one', 'two',
    'three', 'four');
    for elem in 1 .. var_table_varchar.count
    loop
      dbms_output.put_line(elem || ': ' ||
      var_table_varchar(elem));
    end loop;
end;
```

Example : Varray

```
declare
   type varray_varchar is varying array(20) of
   varchar2(10); var_varray_varchar  varray_varchar;
begin
   var_varray_varchar := varray_varchar ('one', 'two',
   'three', 'four');
   for elem in 1 .. var_varray_varchar.count
   loop
         dbms_output.put_line(elem || ': ' ||
         var_varray_varchar(elem));
   end loop;
end;
```

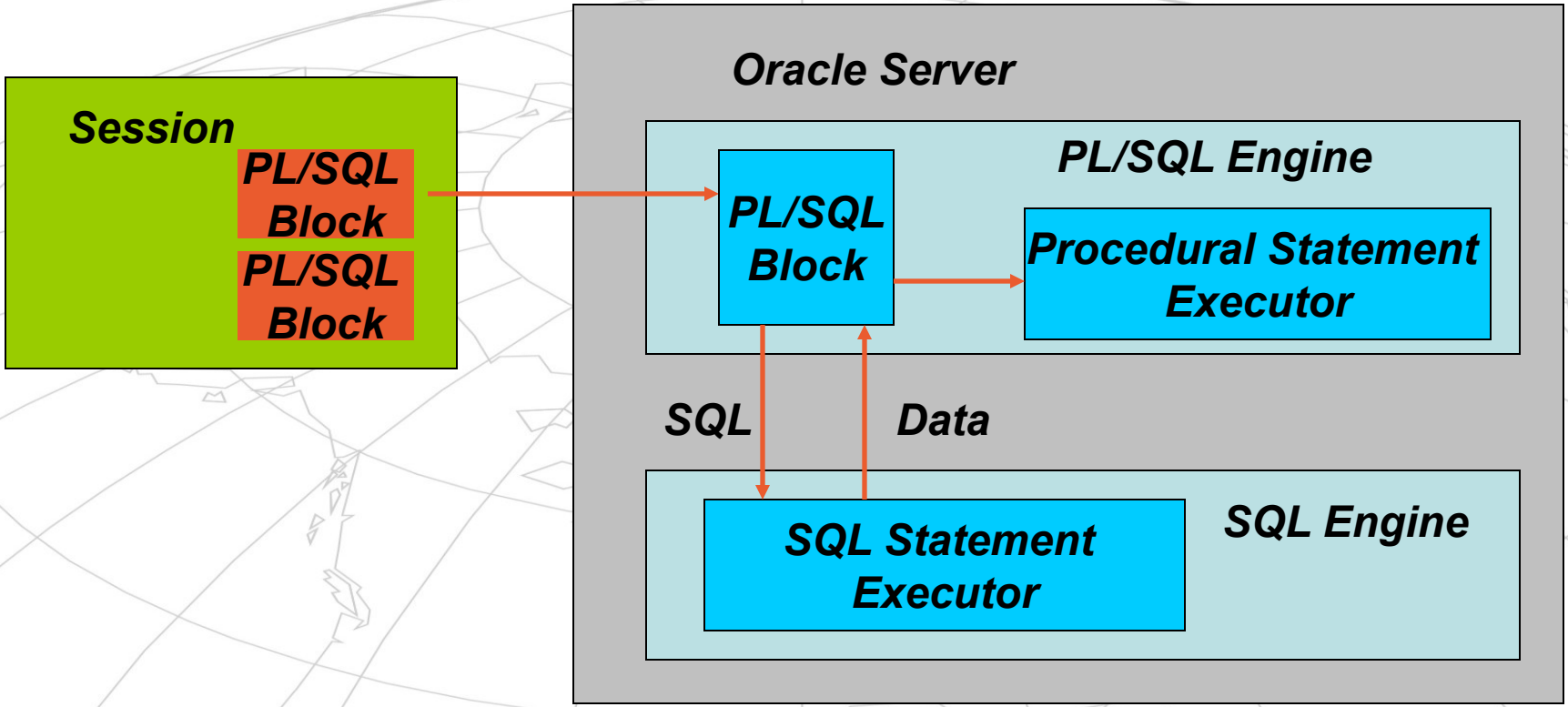
Bulk Processing

- **Supercharge your PL/SQL code with BULK COLLECT and FORALL**
- **Working at a table-level instead of the row-level**
- **Simple and easy to use**

PL/SQL Code

- Consists of two types of statements
  - Procedural (declare, begin, if, while, for …)
  - SQL (select, insert, update, delete)
- Oracle has two engines to process that information
  - PL/SQL Engine
  - SQL Engine
- A Content Switch occurs each time the PL/SQL engine needs to execute a SQL statement
- Switches are fast but large loops can cause performance delays

Context Switches



Bulk collection Categories

- **SELECT or FETCH statements**
  BULK COLLECT INTO
- **Out-Bind binding**
  RETURNING clause
- **In-Bind binding**
  FORALL – INSERT, UPDATE, DELETE

SELECT / FETCH statements

Data may be Bulk Collected/Fetched into:

- **Table.column%TYPE**
- **Record of arrays**
- **Table%ROWTYPE**
- **Cursor%ROWTYPE**
- **Array of records**
- **Nested tables**

Products Table

Create Table:

```
create table products (
        product_id number,
        product_name varchar2(15),
        effective_date date );
```

Insert 1,00,000 records:

```
begin
 -- inserting 100000 records into the products table
  for i in 1 .. 100000
  loop
     insert into products values (i, 'PROD'||to_char(i),sysdate-1);
 end loop;
 commit;
end;
```

Bulk Collect Clause

- Used in a SELECT statement
- Binds the result set of the query to a collection
- Much less communication between the PL/SQL and SQL engines
- All variables in the INTO clause must be a collection

Example : Bulk Collect

```
DECLARE
    TYPE prod_tab IS TABLE OF products%ROWTYPE;
    products_tab                    prod_tab := prod_tab();
    start_time              number;
    end_time                number;
BEGIN
    start_time := DBMS_UTILITY.get_time;
    FOR prod_rec in (SELECT * FROM products
     WHERE effective_date BETWEEN sysdate - 2 AND TRUNC(sysdate))
    LOOP
        products_tab.extend;
        products_tab(products_tab.last) := prod_rec;
    END LOOP;
    end_time := DBMS_UTILITY.get_time;
    DBMS_OUTPUT.PUT_LINE('Conventional ('||products_tab.count||'):
       '||to_char(end_time-start_time));

    Start_time := DBMS_UTILITY.get_time;
    SELECT *
     BULK COLLECT INTO products_tab
    FROM products
    WHERE effective_date BETWEEN sysdate - 2 AND TRUNC(sysdate);
    end_time := DBMS_UTILITY.get_time;
    DBMS_OUTPUT.PUT_LINE('Bulk Collect ('||products_tab.count||'):
       '||to_char(end_time-start_time));
END;
```

Example : Bulk Collect Explicit Cursor Fetch

```
DECLARE
     TYPE prod_tab IS TABLE OF products%ROWTYPE;
     products_tab            prod_tab := prod_tab();
     start_time              number;
     end_time                number;
     CURSOR products_data IS SELECT * FROM products;
BEGIN
     start_time := DBMS_UTILITY.get_time;
     OPEN products_data;
     LOOP
          products_tab.extend;
          FETCH products_data INTO products_tab(products_tab.last);
          IF products_data%NOTFOUND THEN
              products_tab.delete(products_tab.last);
             EXIT;
          END IF;
     END LOOP;
     CLOSE products_data;
     end_time := DBMS_UTILITY.get_time;
     DBMS_OUTPUT.PUT_LINE('Conventional ('||products_tab.count||'): '||to_char(end_time-
        start_time));
     Start_time := DBMS_UTILITY.get_time;
     OPEN products_data;
     FETCH products_data BULK COLLECT INTO products_tab;
     CLOSE products_data;
     end_time := DBMS_UTILITY.get_time;
     DBMS_OUTPUT.PUT_LINE('Bulk Collect ('||products_tab.count||'): '||to_char(end_time-
        start_time));
END;
```

Bulk Collect – FOR ALL

- **We have seen BULK COLLECT with the SELECT statements**
- **For the INSERT, UPDATE and DELETE statements there is the FORALL statement**
- **This is referred to as IN-BINDING**

Example : Bulk Collect – FOR ALL

```
DECLARE
TYPE prod_tab IS TABLE OF products%ROWTYPE;
products_tab   prod_tab := prod_tab();
start_time  number;  end_time   number;
BEGIN
-- Populate a collection - 100000 rows
SELECT * BULK COLLECT INTO products_tab FROM products;

EXECUTE IMMEDIATE 'TRUNCATE TABLE products';
Start_time := DBMS_UTILITY.get_time;
FOR i in products_tab.first .. products_tab.last LOOP
 INSERT INTO products
   VALUES products_tab(i);
END LOOP;
end_time := DBMS_UTILITY.get_time;
DBMS_OUTPUT.PUT_LINE('Conventional Insert: '||to_char(end_time-start_time));

EXECUTE IMMEDIATE 'TRUNCATE TABLE products';
Start_time := DBMS_UTILITY.get_time;
FORALL i in products_tab.first .. products_tab.last
 INSERT INTO products VALUES products_tab(i);
end_time := DBMS_UTILITY.get_time;
DBMS_OUTPUT.PUT_LINE('Bulk Insert: '||to_char(end_time-start_time));
COMMIT;
END;
```

Bulk Collect – Fine Points

- **Use bulk bind techniques for recurring SQL statements in a PL/SQL loop.**
- **Bulk bind rules:**
  - **Can be used with any type of collection**
  - **Collections should be densely filled**
  - **If error, statement is rolled back.**
  - **Prior successful DML statements are not rolled back.**
- **Bulk Collects**
  - **Can be used with implicit or explicit cursors**
  - **Collection is always filled sequentially starting with 1**

Summary

In this session you should have learned the following :

- PL/SQL Collection Types
- Working with:
  - Index By Table
  - Nested Table
  - VARRAY
- Improving code performance with:
  - BULK COLLECT
  - FOR ALL Clauses

Thank You !